2000

Paving the Road to Business Process Administration

P. Rittgen
University of Koblenz - Landau

search

Paving the Road to Business Process Automation

P. Rittgen
University Koblenz-Landau
Rheinau 1
Koblenz, 56075 Germany

***Abstract*- Event-driven Process Chains (EPCs) have been helped to achieve an important role in business process modeling by the commercial success of SAP and ARIS. Both users and IT experts may describe the process to be modelled from their individual perspectives. Event-driven Process Chains, therefore, create a common platform for communication and the analysis of ideas beyond the boundaries of both application and information-system domains. This is accomplished by a semiformal semantics, which gives the participants greater freedom of expression but leads to unintended ambiguities clearly undesirable in later stages of development such as design and implementation. In the literature, several approaches to this problem have been suggested including definitions of a formal semantics for EPCs. We investigate difficulties with such approaches and suggest two solutions: the introduction of a new logical connector (XORAND) and a slight modification of the OR join. This facilitates the design of correct EPCs while continuing to allow freedom of expression, thus enabling a smoother transition into the more formal phases of software development such as design and implementation. A comparative experiment validates these results.**

I. Introduction

Business processes have been studied intensely in the MIS field during this decade. Numerous contributions to the literature have been made by researchers all over the world, making a thorough understanding of the matter difficult. Nevertheless, one approach to business process modeling prevails in practice: the Event-driven Process Chain (EPC) of the ARchitecture of integrated Information Systems (ARIS) [1]. The reasons for this are manifold: on the one hand, the ARIS-toolset for designing EPCs has been available for quite some time providing a hands-on commercial tool for the practitioner; in addition to that the success of the SAP suite of standard business applications helped with the dissemination of this method. On the other hand EPCs have also been studied thoroughly by researchers, as we will show in the following sections.

Nevertheless, there is still some argument concerning the suitability of EPCs for modeling business processes. Advantages such as being highly flexible and easy to learn and understand are compensated by significant disadvantages: first of all ambiguity and vagueness. It cannot be in the interest of the user if the processes described in the specification are interpreted differently by the designer. When this misunderstanding is discovered by the user it is often too late to correct the design accordingly. Where is the path that leads out of this dilemma and towards a better understanding between the participants in the software development process?

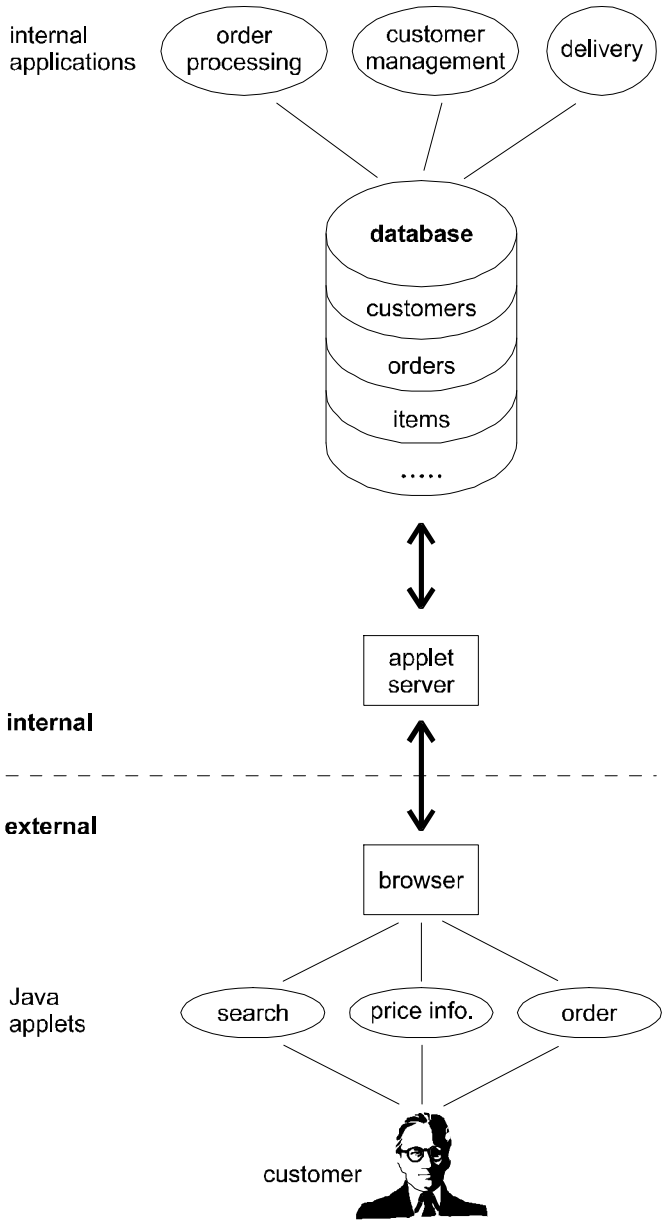

Fig. 1: Architecture of a web-accessible information system

To illustrate the problems we consider the example of a young mail-order company trading software products. Up to now, they were organized in a more or less conventional

way: customers ordered via phone, fax or surface mail. Orders were processed manually and then entered into a database. The products were stocked in physical form as CD-ROMs. Stock management was done with the help of a stand-alone system. Delivery was effected by conventional posting, payment by cheque, credit card or money order. Now this company plans to operate over the internet. Apart from offering new services (such as ordering via the world wide web and downloading of the ordered product), this also requires substantial reorganization: e.g. the isolated information systems for ordering and stocking have to be integrated to allow the potential customer a combined search for price and availability of a product. The head of IT is therefore asked to draw up a sketch of the principal architecture of the new system (see fig. 1). It consists of a central database containing information about customers, orders, items and so on. All applications, internal and external, operate on this database. Internal applications are the ones used only by the staff of the company, such as order and customer management, delivery etc. The external applications can also be accessed by the (potential) customers. They are made accessible to the world by an applet server feeding the user's browser.

To build such a system we have to represent the business processes in a way that allows for their automation. EPCs are a promising candidate for this task because they offer a compromise between general understandability by both domain and IT experts and formalization [5]. An EPC is a graph that consists of events (hexagons) and functions (rectangles) and control flows between them. Connectors split a flow into concurrent (AND), alternative (OR) or strictly alternative (XOR) executions and can also join them (see fig. 6).

Section 2 investigates approaches to a (formal) semantics of EPCs indicating problems with existing approaches, such as the OR join or syntactical restrictions. One solution, a slight modification of the semantics of the OR join, is suggested in section 2.3 assigning a unique interpretation to each EPC. This means that the semantics then fits the prevailing syntax, allowing the simulation of process models. The second solution works the other way round: if we restrict ourselves to the common denominator of all approaches we arrive at a new class of EPCs called hierarchical EPCs (section 3). Finally we discuss the consequences of these suggestions for practical modeling purposes.

II. Semantic Models of EPCs

Since the EPCs were introduced by Scheer there have been many opinions on how a correct EPC should look like. Proposals ranged from syntactical issues (which nodes can be linked to each other) to semantics (what is the exact meaning of a connector?). On the syntactical level some rules have been established that are now generally accepted, for example [2]: An EPC consists of strictly alternating sequences of events and functions (i.e. processes) that are linked by logical connectors (AND, OR, XOR). There are opening connectors (splits) and closing connectors (joins). Among the syntactical restrictions are:

K1: There are no isolated nodes.

K3/4: Functions and events have exactly one incoming and one outgoing edge (except start and end events).

K6: Connectors are either splits (1 input, several outputs) or joins (several inputs, 1 output).

K8/9: An event is always followed by a function and vice versa (modulo connectors).

Sometimes it is also requested that an event should not be followed by an XOR split because events cannot make decisions.

But there is considerably less unanimity on the subject of semantics. Here we sketch only two of the existing approaches: The first was suggested by Scheer himself (together with Chen). That is why we call it the original semantics although it covers only a subset of all EPCs (see section A). A more elaborate model was given later by Langner et al. But it still requires the transformation of an arbitrary EPC into a well-formed one (see section B). Therefore we introduce a new semantics in section C, the so-called XORAND semantics, which is applicable to any EPC. To facilitate the design of correct EPCs we also slightly modify the syntax concerning the problematic OR join.

A. *The Original Semantics*

Only two years after he had introduced EPCs, Scheer himself gave them a formal semantics in a paper he wrote with Chen [3]. The semantics is based on Petri nets, more precisely place/transition nets, which obviously closely resemble EPCs: the functions correspond to transitions, events can be represented by places. The XOR split and join are described by the modules in fig. 2.

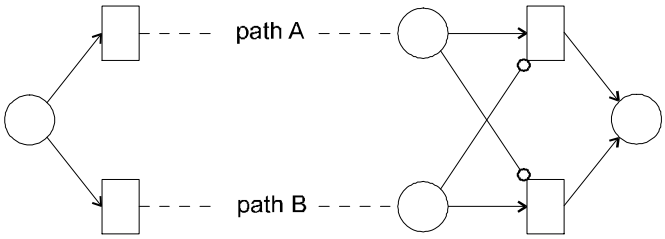

Fig. 2: Petri nets for XOR split and join

The left module is the XOR split where on arrival of a token only one transition can fire removing the token necessary for the other transition to fire. Hence only one path can be activated at a time. The right module represents the XOR join which only fires if not more than one place is marked. Should both places hold tokens, the connector blocks (deadlock), thus indicating a possibly wrongful design of the EPC. This is achieved by the inhibitor edges (the ones with

the small circles at the end) which inhibit firing in the presence of a token.

Analogously Petri-net modules for the AND connectors can be specified (see fig. 3).

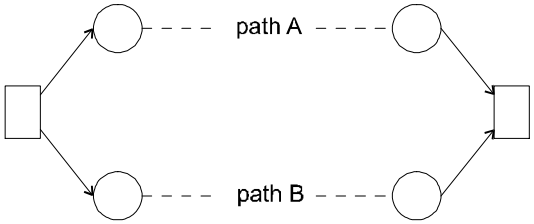

Fig. 3: Petri nets for AND split and join

If we try to do the same for the OR connectors we discover that here the semantics of the join cannot be determined on itself. The EPC on the left side of fig. 4, for example, has a unique interpretation because the join brings together again exactly the paths separated by the split. So the join simply waits for the completion of all paths activated by the split.

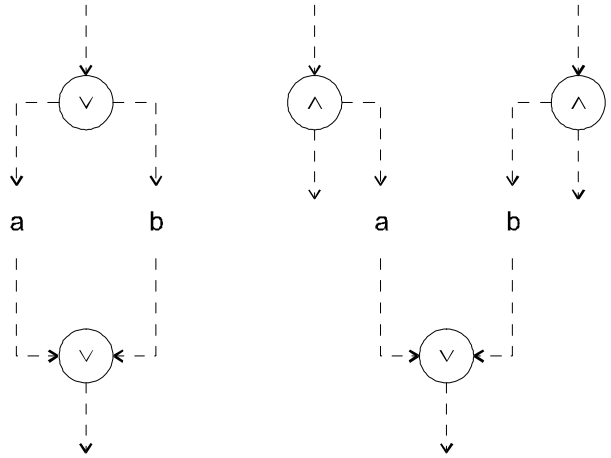

Fig. 4: OR join with and without corresponding split

But what is the meaning of the EPC on the right? According to the semantics of [3] yet to be presented it has no meaning at all because the OR join has no corresponding split. Due to [4] (explained below) the OR join is interpreted as an AND, i.e. it waits for both paths. But perhaps the modeler intended the join to be trig gered by the first completed path. So there are at least three possible interpretations, a situation most probably provoking mistakes in later stages of software development. For this reason we suggest an unambiguous semantics in section C and modify the syntax accordingly. But before that we sketch the OR semantics of [3] and [4].

In [3] there are different tokens for the branches of an OR e.g. token “a” for path A and token “b” for path B. The split informs the join of the to kens to be expected. In fig. 5 the split is to activate both paths and hence the first transition puts both a and b tokens on both successor places. The first two travel along their respective path, the other two tell the join to wait for the travelling to ken from both path A and path B.

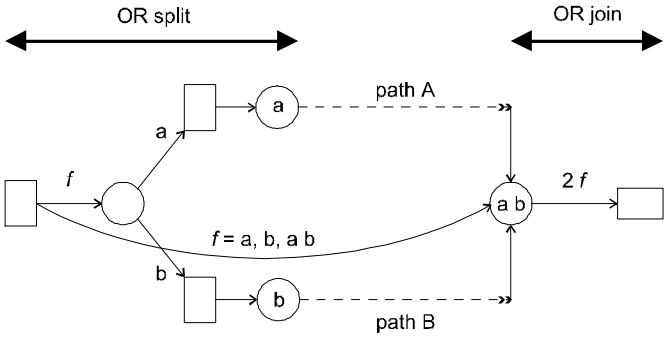

Fig. 5: Petri net for the OR connectors according to [3]

But unfortunately this approach limits the amount of interpretable EPCs severely. It forces the modeler to specify splits and joins correspondingly. This is clearly undesirable for the early phase of analysis where the ideas of the modeler are not yet well structured.

B. A Semantics for Well-formed EPCs

A less restrictive semantics is given by Langner, Schneider and Wehler in [4]. They us e boolean Petri nets with tokens 0 (false or inactive) and 1 (true or active). The OR problem is solved by the simple trick of sending tokens along all paths: a 1 to activate it and a 0 to deactivate it. Now the OR join can wait for the arrival of tokens from all incoming paths and if at least one 1 token is present it activates its successor. The boolean transition is called branch/fork for the OR split and merge/join for the OR join. The opening and closing XOR transitions are branch and merge respectivel y. In the case of the AND they are referred to as fork and join. The firing rules are given by the standard truth tables of propositional calculus with the following exceptions: the entries “0 1” and “1 0” of the AND are not applicable, and neither is the combination “1 1” of XOR. The corresponding joins block on this input instead of passing on a 0 to the successor.

Strictly speaking this semantics only applies to well - formed EPCs. An EPC is well-formed if all generated tokens are extinguished eventually, no dead paths exist and no connector blocks. This is the case if all branches of a split come together in one corresponding join without jumps into or from the branches. Well-formedness is checked by a static and a dynamic analysis only after the transform ation of the EPC into a boolean net. This process involves the restructuring of not-well-formed nets to meet the criteria. The result is always a well-formed net but one that in general has not the same meaning as the EPC from which we started. An example for this is shown in fig. 6. While in the original EPC a delivery is only entered after a possible correction, the corresponding boolean net changes this process so that a delivery is always entered at once (possibly wrongly) and corrected later (if necessary). Other changes include the

removal of connectors K7 and K11 and turning an OR join into an XOR to make it match with the XOR split.

Whether these fundamental changes are admissible can only be judged by the people from the responsible department (accounting in our example). But they are usually not in a position to handle the complex transformations into well-formed nets. Hence problems of this kind can only be solved by a team of IT specialists and users but such a process is rather costly. From an economic point of view we should therefore avoid making EPCs well -formed.

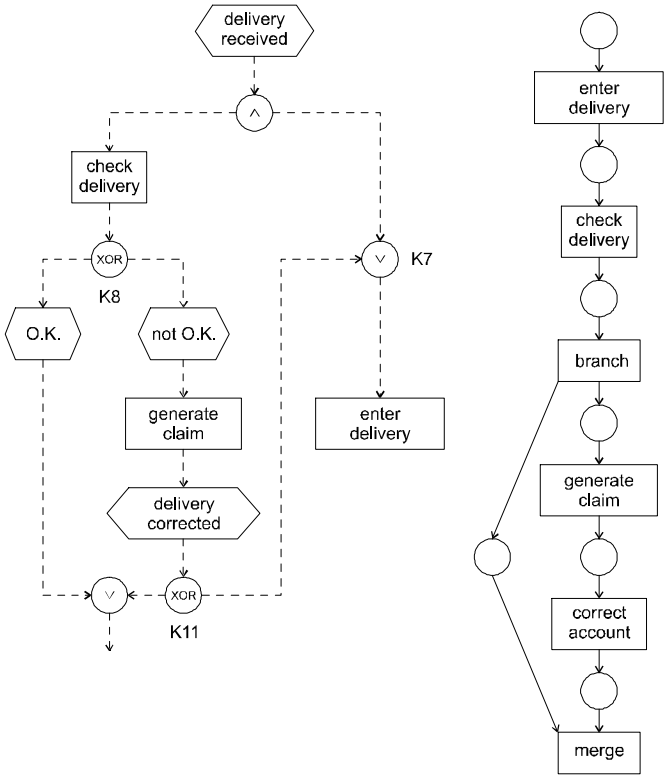

Fig. 6: EPC and boolean net according to [4], figs. 1 & 3

C. XORAND Semantics

In order to enable a simple and unambiguous interpretation of any EPC we have to decouple the meaning of the OR join from that of the OR split. A Petri net for the latter can be given easily (see fig. 7).

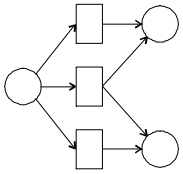

Fig. 7: Petri net for the OR split

But the OR join poses a serious problem because we conventionally assume that it has to wait for all paths activated by the matching split. But what happens if that does not exist? To cover these cases the join has to decide on its own how many tokens it waits for. This corresponds to the meaning of the OR in prop ositional calculus. So if we define the OR in terms of AND and XOR we get:

$E = F_1 \quad F_2$ is equivalent to

$E = E_1$ xor E_2 where $E_1 = F_1$ xor F_2 and $E_2 = F_1 \quad F_2$.

The equivalent EPCs are drawn in fig. 8.

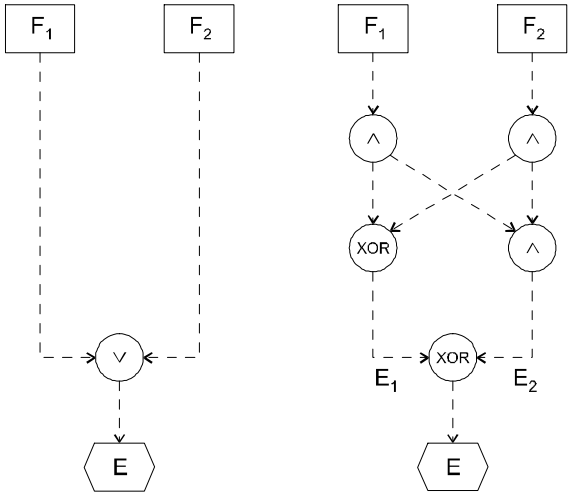

Fig. 8: OR join and its equivalent

On the top level a decision between E_1 and E_2 has to be made. If E_1 is chosen, the two incoming paths are treated as alternatives; so we wait for only one token. But if E_2 is selected the paths are concurrent and we wait for two tokens. Due to the choice between XOR and AND we call it the XORAND semantics. The corresponding Petri net is shown in fig. 9.

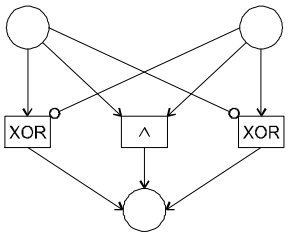

Fig. 9: Petri net for the OR join

To make a decision between E_1 and E_2 there is generally not enough informati on available in the EPC. But the programmer implementing the EPC has to make this decision explicit. What can he do? He might either “guess” the information from the context (running the risk of an error) or ask the user. A better approach might be to equip the EPC model with the missing information. This can be done in one of two ways:

adding the information in textual form as a comment flag to the connector (see fig. 10) or

making the structure of the OR (i.e. the choice between XOR and AND) visible emplo ying a new connector (XORAND).

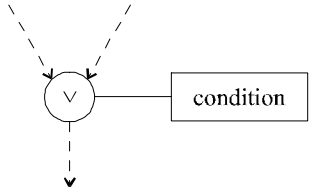

Fig. 10: OR join with condition

Fig. 11 gives two examples for annotated OR joins. The left EPC contains a join that has no matching split. Such a situation can arise in the context of start e vents, for example. Here the comment indicates that the events trigger the EPC independent from each other. If there is a matching split (fig. 11, right) the semantics of the join is absolutely clear. In this case it suffices to assign the same identifier (e.g. a number) to both connectors to indicate the match.

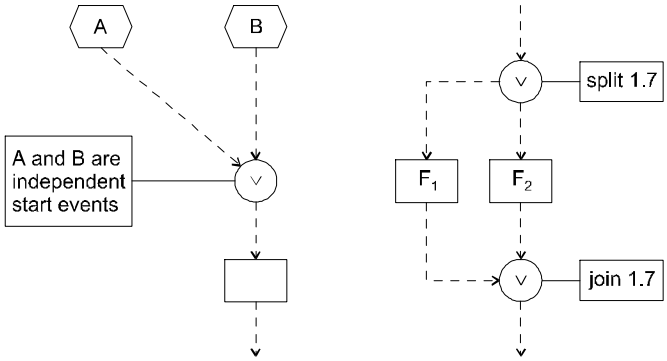

Fig. 11: Examples of annotated OR connectors

The second way of avoiding confusion is the introduction of a new connector that makes the implicit choice between XOR and AND explicit. This so -called XORAND connector (see fig. 12) replaces the OR join. It has two outputs: an XOR output (black quarter) and an AND output (white quarter).

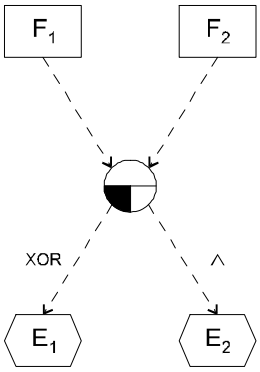

Fig. 12: XORAND connector

This urges the modeler to think about what to do with the two outputs, i.e. what should happen if both paths were completed (E_2) and what if only one was finished (E_1). If you join both outputs with an additional XOR we get the conventional OR join but in this way it is immediately apparent to the modeler that a decision is involved which probably calls for additional information.

Apart from the ambiguous OR there is another problem with the syntax. Especially on higher levels of management EPCs are considered too formal and rigid [5]. Above all they find it hard to identify the events syntactically required between the functions when modeling on an abstract level. We suggest dropping this artificial requirement which was introduced mainly to make EPCs come closer to Petri nets which are also bipartite graphs. But even in higher -level Petri nets it is usually allowed to connect transitions immediately, meaning that there is only one place between them. So, although viewed from a theoretical perspective every function is certainly trig gered by some event, it can do no harm to omit it and there are good reasons to do so for practical purposes. Such missing events can always be generated later if that should prove to be necessary (e.g. the event “invoice entered” after the function “enter invoice” in fig. 13). For similar reasons we might also want to abstract from some of the functions between events.

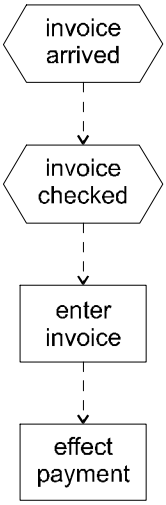

Fig. 13: Non-standard EPC sequence

Consider, for example, the EPC of fig. 13. After the arrival of an invoice it is checked. But if this process is performed manually we are perhaps not interested in it when modeling an information system. So we leave it out. But we do need the result, the checked invoice, to trigger the process of entering the invoice follow ed by the automatic payment. Assuming that both processes are performed by a single program without external intervention there is no need for an event “invoice entered” in between. So we also omit this artificial event.

III. Experiment

We conducted a student experiment in the course of an MIS class on process modeling. The objective of this

experiment was to compare the quality of information systems built with the help of conventional EPCs and EPCs with the modified syntax suggested in C. The class consisted of 20 students divided into two groups A and B. A group had 5 teams of 2 students. The experiment proceeded in 3 phases of 90 minutes each:

Phase 1: Each team had the task to model a core process of a hotel such as reservation, check-in, check-out and purchase. Group A used conventional EPCs, group B the modified syntax. A verbal specification of the respective process was given: "To process an invoice it has to be checked entered, payed and perhaps claimed. The check involves th e ordered quantities, quality and the like. Payment is only effected after a positive check." etc. The specification was incomplete and imprecise to resemble a 'real' one. It left enough room for interpretation and gave only a partial order on the events a nd functions. Consequently, no two EPCs delivered were the same.

Phase 2: Each team "implemented" its own model. As a target language Petri nets were chosen to avoid the intricacies of programming languages. The formality of Petri nets was sufficient to achieve the goal of this phase: to remove any ambiguity present in the EPC and to make explicit the information present in both the model and the modeler's head.

Phase 3: Each team implemented a model of *another* team, a model of a process different from the one it designed in phase 1 to avoid an influence of the own ideas on the interpretation of the other team's model. Again the result is a Petri net. The goal of this phase is to make explicit only the information present in the model.

From this follows that the difference between the two Petri nets for a model consists of the information added by the respective implementation team and not present in the model. The inverse measure, the congruence of the two nets, therefore represents the percentage of informa tion contributed by the model: the higher the congruence the clearer the model. To measure the congruence we proceeded as follows:

First we determined the node congruence by counting the coinciding nodes in both models, i.e. nodes labeled with the same event or function, and relating this to the total number of nodes in both nets. Then we computed the edge congruence for the subset of coinciding nodes in the same way. The overall congruence is the product of the node and edge congruences.

- node congruence = 2 coinciding nodes / node total
- edge congruence = 2 coinciding edges / edge total
- congruence = node congruence edge congruence

For example, if we have two Petri nets, one with 50 and the other with 60 nodes, and the nets share 33 nodes we get a node congruence of 2 33 (the shared nodes are present in both nets) divided by 110, i.e. 60%. If we further assume that 75% of the edges between the 33 shared nodes agree the overall congruence is 45%. Fig. 14 shows the overall congruence of the two nets for each EPC model of the experiment.

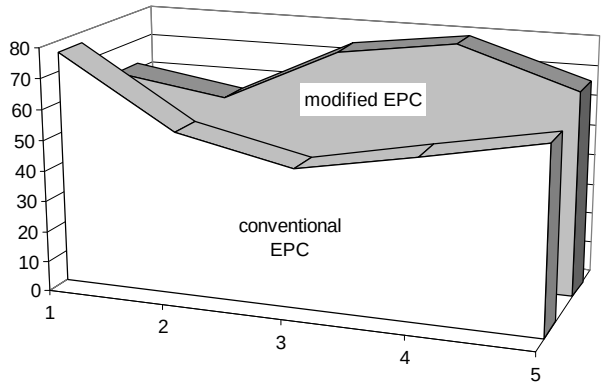

Fig. 14: Net congruences for conventional and modified EPCs

The results indicate that in general a higher congruence can be achieved on the basis of the modified EPCs. The only exception was delivered by team 1 of group A. Although using the conventional, more ambiguous EPC they achieved a congruence of 76.2%, a value well above the average and the second best score of all teams. When looking into the reasons for this exceptional result we found that the corresponding team drew a very simple EPC consisting of only 37 nodes (the others were between 100 and 200).

All in all the experiment allows the (cautious) conclusion that the modifications suggested in section II lead to fewer misinterpretations of the model thus improving the agreement between model and implementation. Hence they facilitate a smooth transition from the analysis phase to later phases speeding up the automation of business processes. We are currently planning a field study to verify these results in a practical setting of larger dimensions.

IV. Conclusion and Outlook

The aim of our paper is to facilitate the automation of business processes by providing a smooth transition from the less structured models of the analysis phase to the more formal ones required in later phases. We investigate the suitability of a typical semiformal business process model, the Event-driven Process Chain, to achieve this aim: can an EPC serve as the starting point of software development? In section II, we identify the major problems with this approach: the ambiguous semantics and the restrictive syntax of EPCs. To remedy the first problem we suggest an unambiguous (XORAND) semantics which includes min or syntactical changes such as an extended OR join and a new XORAND

join. The second problem can be solved by making the syntax less rigid, i.e. by simply allowing to abstract from events or even functions not relevant to the design of the business process. Section III shows the validity and usefulness of these changes in a student experiment. Further experiments should be carried out to verify these results.

The method outlined so far considers only the dynamical aspects of an information system. To complete the description of the system, the business objects manipulated by the process have to be included: documents, data and other resources. An approach to enhance EPCs by object-oriented concepts is suggested in [6]. But modeling the information system alone is not sufficient because it is embedded in the larger system of the enterprise. So apart from the IS level we must take into account the organizational and strategic levels. A so-called multi-perspective approach to enterprise modeling is sketched in [7]. On each level the enterprise is modeled from four perspectives: structure, process, resources and goals. A major challenge of future research is the integration of these views across the level and perspective boundaries.

REFERENCES

[1] Scheer, A.-W.: Architektur integrierter Informationssysteme, Springer, Berlin, 1992

[2] Keller, G.; Teufel, Th.: SAP R/3 prozessorientiert anwenden – iteratives Prozess-Prototyping zur Bildung von Wertschöpfungsketten, Addison-Wesley, Bonn, 1997.

[3] Chen, R.; Scheer, A.-W.: Modellierung von Prozessketten mittels Petri-Netz-Theorie, IWi-Heft 107, Institut für Wirtschaftsinformatik, Universität Saarbrücken, 1994.

[4] Langner, P.; Schneider, Ch.; Wehler, J.: Prozessmodellierung mit Ereignisgesteuerten Prozessketten (EPKs) und Petri-Netzen, WIRTSCHAFTSINFORMATIK, 39 (1997) 5, S. 479-489.

[5] Speck, M.: Akzeptanz und Operationalität von EPK in der Modellierungspraxis – Erfahrungsbericht aus einem Reengineering-Projekt, Arbeitskreistreffen Formalisierung der EPK, Münster, http://www-is.informatik.uni-oldenburg.de/~epk/treffen_170398/speck.ps, March 17, 1998

[6] Rittgen, P.: From Process Model to Electronic Business Process, in Avison, D; Christiaanse, E; Ciborra, C.U; Kautz, K; Pries-Heje, J; Valor, J: Proceedings of the 7th European Conference on Information Systems (ECIS), 23-25 June, 1999, Copenhagen Business School, Denmark.

[7] Frank, U.: Enriching Object-Oriented Methods with Domain Specific Knowledge: Outline of a Method for Enterprise Modeling. Arbeitsberichte des Instituts für Wirtschaftsinformatik Nr. 4, Universität Koblenz-Landau, 1997.